Orchestrating Computation and Physical Dynamics: Response to RFI for NITRD

August 21, 2008

This response considers the orchestration of computing with physical processes. It argues that to realize its full potential, the core abstractions of computing need to be rethought to incorporate essential properties of physical dynamics. All branches of computing, networking, and systems theory are affected, resulting in a research agenda that can only be effectively pursued as a national or international effort.

Most microprocessors today are embedded in systems that are not first-and-foremost computers. They are cars, medical devices, instruments, communication systems, industrial robots, toys, games, etc. Key to these microprocessors is their interaction with physical processes through sensors and actuators. Such microprocessors increasingly resemble general-purpose computers. They are becoming networked and intelligent, often at the cost of reliability. At the same time, general-purpose computers are increasingly being asked to perform complex interactions with physical processes. They integrate media such as video and audio, and through the migration to handheld platforms and pervasive computing, sense physical dynamics and control physical devices.

The foundations of computing do not support this mission well. In the Turing-Church abstraction, computation is about the transformation of data, not about physical dynamics. This abstraction deliberately omits the passage of time, an essential property of physical dynamics.

Computer scientists and engineers have heavily exploited this omission. The central fact is that "correct" execution of nearly any computer program has nothing to do with how long it takes to do anything. Engineers have to step outside the programming abstractions to specify timing properties. Timing needs to become a correctness property rather than a quality of service measure. This requires profound changes in computing and networking.

Computers have become so fast that surely the passage time in most physical processes should be able to be handled without special effort. But then why is the latency of audio signals in modern PCs as large as it was 20 years ago? Audio processes are quite slow by physical standards, and a latency of a large fraction of a second is enormous. To achieve good audio performance in a computer (e.g. in a set-top box, which is required to have good audio performance), engineers are forced to discard many of the innovations of the last 30 years of computing. They often work without an operating system, without virtual memory, without high-level programming languages, and without memory management, and they use microprocessors without caches, dynamic dispatch, or speculative execution. Those innovations are built on the key premise that time is irrelevant to correctness. By contrast, what these systems need is not faster computing, but physical actions taken at the right time. It needs to be a semantic property, not a quality factor.

But surely the "right time" is expecting too much, the reader may object. The physical world is neither precise nor reliable, so why should we demand this of computing systems? Instead, we must make the systems robust and adaptive, building reliable systems out of unreliable components. Clearly systems need to be designed to be robust, but we should not blithely discard the reliability we have. Electronics technology is astonishingly precise and reliable, more than any other human invention. We routinely deliver circuits that will perform a logical function essentially perfectly, on time, billions of times per second, for years. Shouldn't we exploit this remarkable achievement?

This problem is going to get worse. As embedded systems become more networked and intelligent, their character fundamentally changes. They are no longer black boxes, designed once and immutable in the field. Instead, they are pieces of a larger system, a dance of electronics, networking, and physical processes. An emerging buzzword (that few seem particularly fond of) for such systems is *cyber-physical systems* (CPS). Such systems will unquestionably have an enormous impact on technical dominance.

Applications of CPS have the potential to rival the 20-th century IT revolution. They include high confidence medical devices and systems, assisted living, traffic control and safety, automotive systems, process control, energy conservation, environmental control, avionics, instrumentation, critical infrastructure control (electric power, water resources, and communications systems for example), distributed robotics (telepresence, telemedicine), defense systems, manufacturing, and smart structures. It is easy to envision new capabilities that are technically well within striking distance, but that would be extremely difficult to deploy using today's methods. Consider, for example, a city with no traffic lights, where each car provides the driver with adaptive information on speed limits and clearance to pass through intersections. We have in hand all the technical pieces for such a system, but achieving the requisite level of confidence in the technology seems decades off.

The challenge of integrating computing and physical processes has been recognized for some time, having motivated for example the emergence of hybrid systems theories. These theories blend the dynamical systems models of electrical engineers with automata models of computer scientists. But the effort needs to extend beyond systems theories into the abstraction stack on which engineers build applications. Today's computing and networking technologies *unnecessarily* impede progress towards CPS applications, and dynamical systems theories unnecessarily omit software and network behavior.

The solution pervades the abstraction stack. Beginning bottom-up, computer architects have gone overboard exploiting the irrelevance of timing to achieve better performance. Multi-level caches, dynamic dispatch, speculative execution, and bus architectures are all notable culprits. The research challenge is to achieve comparable performance with predictable and repeatable timing.

Continuing up the stack, there is a long history of attempts to insert timing features into programming languages. These are generally done, however, without fundamental changes in the semantics of the languages, particularly with regard to concurrency. Domain-specific languages with temporal semantics (e.g. Simulink, LabVIEW) have firmly taken hold in some areas, showing that there are alternatives. These are radically different from imperative and functional languages that dominate the programming language community. But they remain outside the mainstream of software engineering, are not well integrated into software engineering processes and tools, and have not benefited from many innovations such as data abstraction and strong type systems.

An attractive alternative to new programming languages is notations that work at the level of tasks or components. These are attractive because they can exploit experience with conventional imperative or functional languages, which can be used to specify detail functionality, and they can embrace models of physical systems. Various innovations in coordination languages and actor models look promising, but require adaptation to express temporal dynamics. I envision an innovation like what C++ did to C, which provided language support for object-oriented design. For example, an actor model with temporal dynamics could be supported by a C++++ that introduced notations for expressing concurrency and timing.

Similar research will be needed in systems theory, semantics, verification, security, operating systems, and networking. The emphasis needs to be on predictable repeatable dynamics rather than on performance optimization. This requires more than incremental improvements, which will, of course, continue to help. But effective orchestration of computing and physical processes requires semantic models that reflect properties of interest in both.